

## 02 | 编码阶段能做什么：秀出好的code style

2020-05-09 罗剑锋

罗剑锋的C++实战笔记

[进入课程 >](#)



讲述：Chrono

时长 12:14 大小 11.21M



你好，我是 Chrono。

上节课我介绍了 C++ 程序的生命周期和编程范式，今天我就接着展开来讲，看看在编码阶段我们能做些什么事情。

在编码阶段，我们的核心任务就是写出在预处理、编译和运行等不同阶段里执行的代码，还记得那句编程格言吗：



“任何人都能写出机器能看懂的代码，但只有优秀的程序员才能写出人能看懂的代码。”

所以，我们在编码阶段的首要目标，不是实现功能，而是要写出清晰易读的代码，也就是要有好的 code style。

怎样才能写出 human readable 的好代码呢？

这就需要有一些明确的、经过实践验证的规则来指导，只要自觉遵守、合理运用这些规则，想把代码写“烂”都很难。

在此，我强烈推荐一份非常棒的 [🔗指南](#)，它来自 OpenResty 的作者章亦春，代码风格参照的是顶级开源产品 Nginx，内容非常详细完善。

不过有一点要注意，这份指南描述的是 C 语言，但对于 C++ 仍然有很好的指导意义，所以接下来我就以它为基础，加上我的工作体会，从**代码格式**、**标识符命名**和**注释**三个方面，讲一下怎么“秀出好的 code style”。

## 空格与空行

当我们拿到一份编码风格指南的时候，不论它是公司内部还是外部的，通常第一感觉就是“头大”，几十个、上百个的条款罗列在一起，规则甚至细致到了标点符号，再配上干巴巴的说明和示例，不花个半天功夫是绝对看不完的。而且，最大的可能是半途而废，成了“从入门到放弃”。

我写了很多年代码，也看过不少这样的文档，我从中总结出了一条最基本、最关键的规则，只要掌握了这条规则，就可以把你 code style 的“颜值”起码提高 80%。

这条“神奇”的规则是什么呢？

认真听，只有五个字：**留白的艺术**。

再多说一点，就是像“写诗”一样去写代码，**恰当地运用空格和空行**。不要为了“节省篇幅”和“紧凑”而把很多语句挤在一起，而是要多用空格分隔开变量与操作符，用空行分隔开代码块，保持适当的阅读节奏。

你可以看看下面的这个示例，这是我从某个实际的项目中摘出来的真实代码（当然，我隐去了一些敏感信息）：

```

1  if(!value.contains("xxx")){
2      LOGIT(WARNING,"value is incomplete.\n")
3      return;
4  }
5  char suffix[16]="xxx";
6  int data_len = 100;
7  if(!value.empty()&&value.contains("tom")){
8      const char* name=value.c_str();
9      for(int i=0;i<MAX_LEN;i++){
10         ... // do something
11     }
12     int count=0;
13     for(int i=0;i<strlen(name);i++){
14         ... // do something
15     }
16 }

```

这段代码真可谓是“高密度”，密密麻麻一大堆，看着“有如滔滔江水，连绵不绝”，读起来让人“窒息”，code style 非常糟糕。

应用“留白的艺术”，代码就变成了下面的样子：

```

1  if (!value.contains("xxx")) {                                // if后{前有空格
2      LOGIT(WARNING, "value is incomplete.\n")                // 逗号后有空格
3      return;                                                  // 逻辑联系紧密就不用加空行
4  }
5                                                                // 新增空行分隔段落
6  char suffix[16] = "xxx";                                    // 等号两边有空格
7  int data_len = 100;                                         // 逻辑联系紧密就不用加空行
8                                                                // 新增空行分隔段落
9  if (!value.empty() && value.contains("tom")) {                // &&两边有空格
10     const char* name = value.c_str();                        // 等号两边有空格
11                                                                // 新增空行分隔段落
12     for(int i = 0; i < MAX_LEN; i++){                          // =;<处有空格
13         ... // do something
14     }
15                                                                // 新增空行分隔段落
16     int count = 0;                                           // 等号两边有空格
17                                                                // 新增空行分隔段落
18     for(int i = 0; i < strlen(name); i++){                    // =;<处有空格
19         ... // do something
20     }
21 }

```

加上了适当的空格和空行后，代码就显得错落有致，舒缓得当，看着就像是莎翁的十四行诗，读起来不那么累，也更容易理清楚代码的逻辑。

我还有个私人观点：**好程序里的空白行至少要占到总行数的 20% 以上**。虽然比较“极端”，但也不失为一个可量化的指标，你可以在今后的实际工作中尝试一下。

## 起个好名字

有了好的代码格式，接下来我们要操心的就是里面的内容了，而其中一个很重要的部分就是为变量、函数、类、项目等起一个好听易懂的名字。

这里有一个广泛流传的笑话：“**缓存失效与命名是计算机科学的两大难题**。”把起名字与缓存失效（也有说是并发）相提并论，足见它有多么重要了，值得引起你的重视。

但其实命名这件事并不难，主要就在于平时的词汇和经验积累，知道在什么情况下用哪个单词最合适，千万不要偷懒用“谜之缩写”和汉语拼音（更有甚者，是汉语拼音的缩写）。由于现在搜索引擎、电子词典都很发达，只要你有足够认真的态度，在网上一搜，就能够找到合适的名字。

另外，你还可以用一些已经在程序员之间形成了普遍共识的变量名，比如用于循环的 i/j/k、用于计数的 count、表示指针的 p/ptr、表示缓冲区的 buf/buffer、表示变化量的 delta、表示总和的 sum.....

关于命名的风格，我所知道的应用比较广的有三种。

第一种风格叫“匈牙利命名法”，在早期的 Windows 上很流行，使用前缀 i/n/sz 等来表示变量的类型，比如 iNum/szName。它把类型信息做了“硬编码”，不适合代码重构和泛型编程，所以目前基本上被淘汰了。

不过它里面有一种做法我还是比较欣赏的，就是给成员变量加“m\_”前缀（member），给全局变量加“g\_”前缀（global），比如 m\_count、g\_total，这样一看就知道了变量的作用域，在大型程序里还是挺有用的。

第二种风格叫“CamelCase”，也就是“驼峰式命名法”，在 Java 语言里非常流行，主张单词首字母大写，比如 MyJobClass、tryToLock，但这种风格在 C++ 世界里的接受程度

不是太高。

第三种风格叫 “snake\_case”，用的是全小写，单词之间用下划线连接。这是 C 和 C++ 主要采用的命名方式，看一下标准库，里面的 vector、unordered\_set、shrink\_to\_fit 都是这样。

那么，你该选用哪种命名风格呢？

我的建议是“取百家之长”，混用这几种中能够让名字辨识度最高的那些优点，就是四条规则：

1. 变量、函数名和名字空间用 snake\_case，全局变量加 “g\_” 前缀；
2. 自定义类名用 CamelCase，成员函数用 snake\_case，成员变量加 “m\_” 前缀；
3. 宏和常量应当全大写，单词之间用下划线连接；
4. 尽量不要用下划线作为变量的前缀或者后缀（比如 \_local、name\_），很难识别。

下面我举几个例子，你一看就能明白了：

 复制代码

```
1  #define  MAX_PATH_LEN  256                //常量，全大写
2
3  int  g_sys_flag;                          // 全局变量，加g_前缀
4
5  namespace linux_sys {                    // 名字空间，全小写
6      void get_rlimit_core();              // 函数，全小写
7  }
8
9  class FilePath final                     // 类名，首字母大写
10 {
11 public:
12     void set_path(const string& str);      // 函数，全小写
13 private:
14     string m_path;                        // 成员变量，m_前缀
15     int    m_level;                      // 成员变量，m_前缀
16 };
17
18
```

命名另一个相关的问题是“名字的长度”，有人喜欢写得长，有人喜欢写得短，我觉得都可以，只要易读易写就行。

不过一个被普遍认可的原则是：**变量 / 函数的名字长度与它的作用域成正比**，也就是说，局部变量 / 函数名可以短一点，而全局变量 / 函数名应该长一点。

想一下，如果你辛辛苦苦起了一个包含四五个单词的长名字，却只能在短短十几行的循环体里使用，岂不是太浪费了？

## 用好注释

写出了有好名字的变量、函数和类还不够，要让其他人能“一眼”看懂代码，还需要加上注释。

“注释”在任何编程语言里都是一项非常重要的功能，甚至在编程语言之外，比如配置文件（ini、yml）、标记语言（html、xml），都有注释。一个突出的反例就是 JSON，没有注释功能让许多人都很不适应。

注释表面上的功能很简单，就是给代码配上额外的文字，起到注解、补充说明的作用。但就像是写文章一样，写些什么、写多写少、写成什么样子，都是大有讲究的。

你可能也有不少写注释的经验了，一般来说，注释可以用来阐述目的、用途、工作原理、注意事项等代码本身无法“自说明”的那些东西。

但要小心，注释必须要正确、清晰、有效，尽量言简意赅、点到为止，不要画蛇添足，更不能写出含糊、错误的注释。

比如说，有这么一个模板函数 `get_value`：

```
1  template<typename T>
2  int get_value(const T& v);
```

 复制代码

代码很简单，但可用的信息太少了，你就可以给它加上作者、功能说明、调用注意事项、可能的返回值，等等，这样看起来就会舒服得多：

```
1 // author   : chrono
2 // date     : 2020-xx-xx
3 // purpose  : get inner counter value of generic T
4 // notice   : T must have xxx member
5 // notice   : return value maybe -1, means xxx, you should xxx
6 template<typename T>
7 int get_value(const T& v);
```

你可能注意到了，在注释里我都用的是英文，因为英文（ASCII，或者说是 UTF-8）的“兼容性”最好，不会由于操作系统、编码的问题变成无法阅读的乱码，而且还能够锻炼自己的英语表达能力。

不过，用英文写注释的同时也对你提出了更高的要求，最基本的是**不要出现低级的语法、拼写错误**。别笑，我就经常见到有人英文水平不佳，或者是“敷衍了事”，写出的都是“Chinglish”，看了让人哭笑不得。

写注释最好也要有一些标准的格式，比如用统一的“标签”来标记作者、参数说明什么的。这方面我觉得你可以参考 Javadoc，它算是一个不错的工程化实践。

对于 C++ 来说，也有一个类似的工具叫 Doxygen，用好它甚至可以直接从源码生成完整的 API 文档。不过我个人觉得，Doxygen 的格式有些过于“死板”，很难严格执行，是否采用就在于你自己了。


除了给代码、函数、类写注释，我还建议**最好在文件的开头写上本文件的注释**，里面有文件的版权声明、更新历史、功能描述，等等。

下面这个就是我比较常用的一个文件头注释，简单明了，你可以参考一下。

```
1 // Copyright (c) 2020 by Chrono
2 //
3 // file   : xxx.cpp
4 // since  : 2020-xx-xx
5 // desc   : ...
```



另外，注释还有一个很有用的功能就是 `todo`，作为功能的占位符，提醒将来的代码维护者（也许就是你），比如：

 复制代码

```
1 // TODO: change it to unordered_map
2 // XXX: fixme later
```

总的来说，要写好注释，你要时刻“换位思考”，设身处地去想别人怎么看你的代码，这样的话，上面的那些细则也就不难实施了。

## 小结

在编码阶段，拥有一个良好的编程习惯和态度是非常重要的（我见过太多对此漫不经心的“老”程序员）。今天，我只介绍了三个最基本的部分，再来“敲黑板”画个重点：

1. 用好空格和空行，多留白，让写代码就像写诗一样；
2. 给变量、函数、类起个好名字，你的代码就成功了一半；
3. 给变量、函数、类再加上注释，让代码自带文档，就成了“人能够看懂的代码”。

有了这个基础，你还可以更进一步，使用其他高级规则写出更好的代码，比如函数体不能太长、入口参数不宜过多，避免使用 `else/switch` 导致层次太深（圈复杂度），等等，这些虽然也很有用但比较琐碎，暂时就不细说了。

对了，还有一招“必杀技”：善用 `code review`，和你周围的同事互相审查代码，可以快速改善自己的 `code style`。

## 课下作业

最后是课下作业时间，给你留两个思考题：

1. 你用过哪些好的 `code style`，你最喜欢今天介绍的哪几条？
2. 注释在代码里通常的作用是“说明文档”，但它还有另外一个重要的用法，你知道吗？



欢迎你在留言区写下你的思考和答案，如果觉得对你有所帮助，也欢迎分享给你的朋友，我们下节课见。

## 课外小贴士

1. 正文里没有重复讲解一些太过于基本的风格指导，比如统一缩进、花括号对齐、80列限制等等，你肯定也对它们有共识，可以课后看OpenResty文档的链接去详细了解。
2. Google C++ Style Guide是另一份著名的代码风格指南，可以在GitHub上找到，但我个人感觉它的“公司色彩”太重。
3. 在《设计模式》中，作者们曾经提到，为模式选择恰当的名字也是难点之一，可见命名的确是个“大学问”。
4. “驼峰式命名法”也分为两种，“大驼峰”(UpperCamelCase)和“小驼峰”(lowerCamelCase)。
5. 网上有很多工具可以检查C++代码风格，一个比较常见的是cpplint。它是一个Python脚本，可以用命令“`sudo pip install cpplint`”安装。



© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 精选留言 (20)

写留言



Carlos

2020-05-09

哈哈, 容我先说一句题外话, 我作为一个 c++ 入门新手, 昨天和前天还真的读了一份 "公司色彩很重" 的 code style guide 😓(虽然没完全读懂).

1. 今天文章中的 "留白", 和 "命名" 我倒是注意到了, 但是可能因为我目前写的 c++ 代码基本上都是不超过 50 行的练习文件, 所以注释这一点我没有注意到, 从现在开始注意. ...

展开 ▾

作者回复:

1.可以多看看一些开源项目, 它们的代码量比较大, 你就可以体会到空格和空行的作用了。  
专栏的GitHub仓库代码虽然比较小, 但也都应用了这些规范, 可以做参考。

2.说的很对, 这也是注释一个非常重要的作用。我的建议是尽量不要删代码, 毕竟是自己辛辛苦苦写出来的, 删掉太可惜了。

应该用注释暂时禁用, 放一段时间, 如果确实没有用再考虑删除。

3.这个是include guard, 后面的预处理阶段就会讲, 你这是提前预习了, 笑。



8



jxon-H

2020-05-09

磨刀不误砍柴工, 编码风格我觉得不仅是软件开发行业的共识, 更是一种软件开发的文  
化。回顾我自己写过的代码, 简直惨不忍睹。罗老师这节内容让我收获极大, 也让进一步加深了以下认识:

1、之所以讲究编码风格, 因为软件的规模越大, 协作性要求就越高, 软件开发是一件群...

展开 ▾

作者回复: 写的太好了, 我非常感同身受。



4



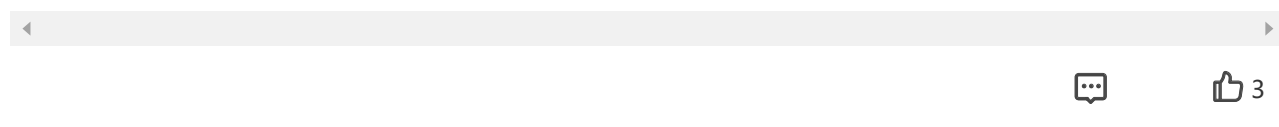
Geek\_0315ca

2020-05-09

我比较喜欢变量名使用m和g前缀，说明变量的作用域范围；todo注释标注自己未实现的功能和想法💡；函数体内部使用空行分离不同的代码片段

展开 ∨

作者回复: 对，这个几个对于改善代码可读性非常有用，而且也简单容易上手。



嵇斌

2020-05-09

分享我的一些实践：

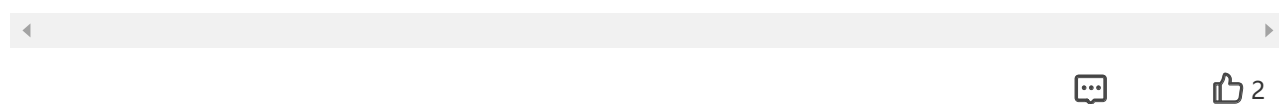
1. 匈牙利命名真心不推荐了，参考《clean code》中关于avoid encoding的说明。
2. 如果是存现代c++工程，不考虑兼容C的话. Header guards可以使用#pragma once替代。
3. 避免注释，最好使用代码来阐述。很多信息可以通过代码仓库来表达，比如commit ...

展开 ∨

作者回复: 分享的经验质量非常高，nice。

关于pragma once，我的建议还是要慎用，因为很多时候我们还会导出纯C接口给外界，还是include guard最保险。

而且我记得在哪里看过资料，好像是cppreference上吧，pragma once也是有缺陷的。

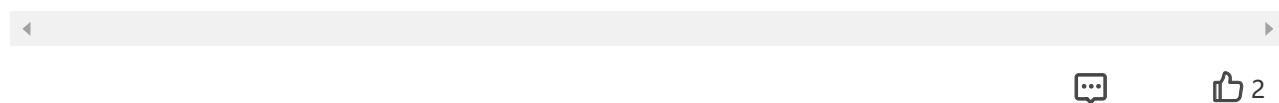


幻境之桥

2020-05-09

在此基础上使用 clang-format 统一并减少大部分手工格式化的工作

作者回复: nice，用clang-format这样的工具能够减少很多手工的工作，但编码风格的意识还是要有的，不能完全依赖工具。



winsummer

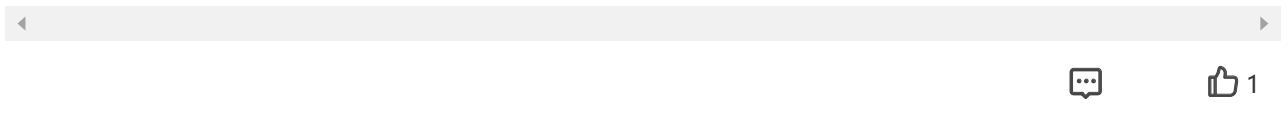
2020-05-09

老师，函数的注释是写在声明处还是写在定义处好呢

展开 ∨

作者回复: 看怎么用, 如果是对外发布的, 就写在声明, 如果是内部用的, 就在定义处。

但一定要注意, 最好不要两处都有注释, 否则一旦有变动, 维护保持同步很麻烦。



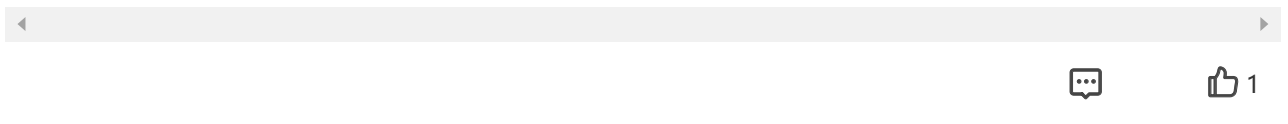
**Yaxe**

2020-05-09

昨天上github看cpp\_study这个仓库的时候 发现头像莫名熟悉, 才知道之前star的注释版nginx也是罗大写的 十分有缘 学习学习!

展开 ∨

作者回复: 笑, 看来我的隐身能力还是挺成功的。



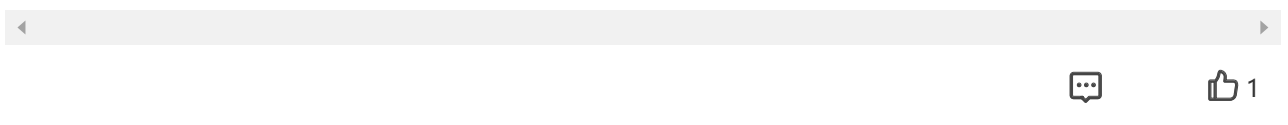
**Sochooligan**

2020-05-09

还没看完, “20%的留白”, 观点非常启发人。

展开 ∨

作者回复: 有这个指标可以强制让大家多用空白, 改善代码的可读性, 具体的数字可以根据实际情况调整。



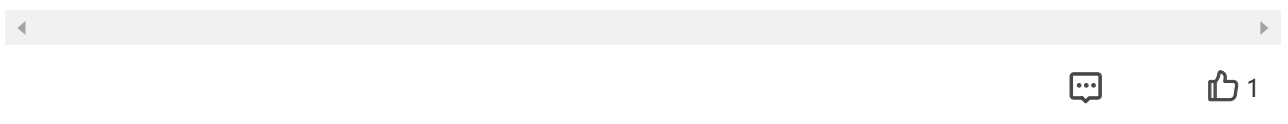
**fl260919784**

2020-05-09

Facebook的命名方式也很赞, 类JAVA风格; 除了函数、变量、类的命名, 还有文件夹、文件、namespace的命名也很重要

展开 ∨

作者回复: 我是标准库、Boost、Nginx看习惯了, 对于CamelCase总是不太适应。

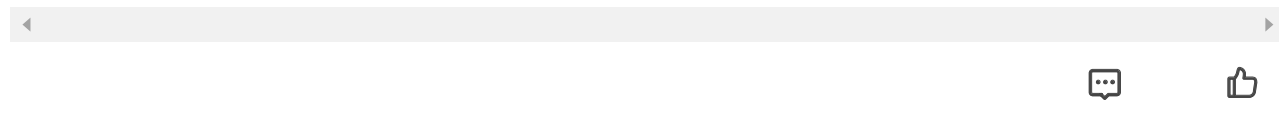


**hb**

2020-05-11

其实各个语言都有自己的code style，例如OC就是习惯驼峰命名，基本不用加 "\_"

作者回复: 很对，但对于C++来说，并没有官方推荐的style，这大概也是C++崇尚的自由吧，我们可以任意选择自己喜欢的style。



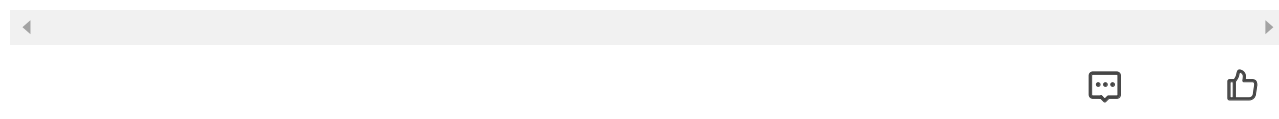
**Geek\_98dc22**

2020-05-10

1. 命名格式 我还是偏爱驼峰式命令法，函数采用大驼峰，变量采用小驼峰；成员变量用m\_，全局用g\_；代码中增加空行的使用值得借鉴。
2. 个人觉得注释不要到处都是，而是类或函数命名无法表达其主要内容或里面有些特殊原因可以增加注释，提高可读性。

展开 ▾

作者回复: 编码风格不是强制的，只要在团队里保持统一就好，其实驼峰式命名法在很多开源项目中都有，可能是跟Java流行有关吧。



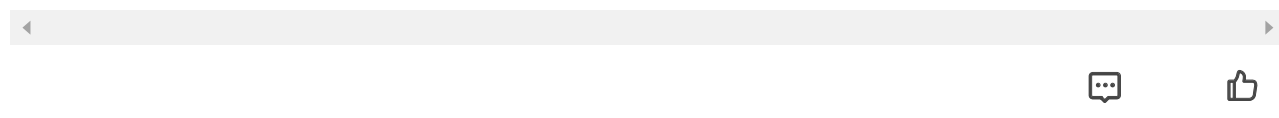
**c1rew**

2020-05-10

代码统一规范，推荐一个工具，astyle，命令行或者vscode也有插件，大家提交代码前一键格式化，风格还可以根据配置自定义。

展开 ▾

作者回复: 好东西欢迎大家一起分享。

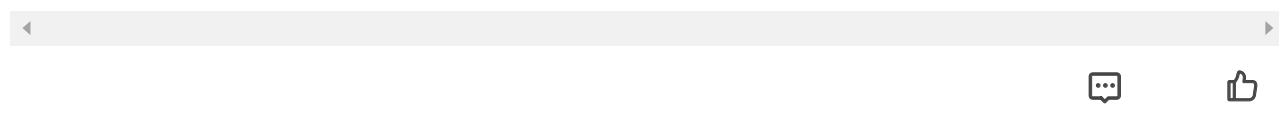


**dao**

2020-05-10

关于编码风格也可以简单用 lint 和 editorconfig 来辅助

作者回复: 有IDE和工具辅助当然很好，但有的时候还是得靠人工，所以编码风格的意识一定要有。





抽抽

2020-05-10

我觉得如果代码自己把自己写清楚了，比如清晰的布局，贴切的函数和变量名，适当的空行分割等等，注释就不必了。这是上策。

如果不行，那尽量写上规整周全的注释，帮助其他人维护。这是中策。

...

展开 ▾

作者回复: 记住代码是写给别人看的，自己看觉得很清楚，可别人不一定理解，所以适当的注释还是要有的。

最基本的文件头、函数和类的功能说明是必须的，关键的业务逻辑部分最好加上注释，每次有重大修改也应该加上注释说明。



土土人

2020-05-10

关于文件名的规范有没有什么建议呀？

展开 ▾

作者回复: 一般文件名都是跟类名相同的，所以做好面向对象设计，分解出好的类就行了。



黄骏

2020-05-10

觉得linux内核的代码注释值得学习，很详细。工作中读了一些开源代码，读注释少的代码会很痛苦，比如，ceph，哈哈

展开 ▾

作者回复: Linux的代码也是很规整的，值得读。注释少的代码看起来确实累，可以用一些工具辅助，比如画uml图。



泡泡龙

2020-05-09

喜欢老师说的写好注释应该换位思考。

还有就是我觉得团队英语水平不行时候还是应该写汉字注释，要比蹩脚的英文注释更好一些。

...

展开 ∨

作者回复:

1.是的，这个也要看团队的整体水平。小段注释尽量用英文，复杂的逻辑我们可能用英文表述不清楚，就可以用中文。

2.git历史当然也可以，但有的时候在代码提交历史里查找也是比较累人的事情，直接放开注释更省事。

3.有多种选择不是坏事，找到最适合自己的就好。



qinsi

2020-05-09

There are 2 hard problems in computer science: cache invalidation, naming things, and off-by-1 errors.

团队英文水平有限，试过直接用中文来命名一些业务上较难翻译成英文或翻译后较繁琐的概念，用作变量名和函数名，反而没那么纠结，代码review或接手一看就懂

展开 ∨

作者回复: 如果是与国情结合比较紧密的业务，没有合适的英文词，那就可以用中文了，但最好配上合适的注释。



Tedeer

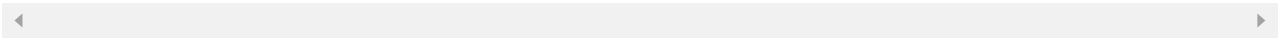
2020-05-09

用过较好的codestyle是《阿里java开发手册》中的介绍，vscode中prettier代码格式化工具，最喜欢今天介绍留白（清爽）和命名，以前在开发中遇到过long a = 10l的情况，被坑了。我觉得注释写的好也是代码协同者的交流语言，节约时间成本。

展开 ∨



作者回复: 对, 编码规范看似简单枯燥, 但却是最应该被重视的, 就像是要求大家都用标准普通话交流。



**wuwei**

2020-05-09

留白的习惯我一直都有。命名倒是有点问题, `member_function`没有加`m_`的习惯。注释只做了代码说明和注释实验代码, 文件开头、函数和类的说明一直没做, 要改改了。

作者回复:

1.注意, 是成员变量加`m_`, 成员函数不需要。

而且加`m_`也不是强制的, 只是我个人的推荐, 你一定要找到对自己最合适的风格。

2.其实真要全写各种注释也是很累的, 看实际情况, 在大中型项目中很有必要, 小项目可以适当省略。

如果有code review, 最好听一下别人的意见, 看他们觉得哪里最需要注释。

